

Zoom meeting ID

- Calendar URL:
<https://bit.ly/3aYQRkg>
 - Please subscribe to this URL now!
- To save bandwidth
 - Turn off your camera
 - Mute your microphone
- Use campuswire for questions

The screenshot shows a calendar for Monday, March 16, 2020. A red circle highlights a meeting event titled "Lecture: language modeling" scheduled for 08:00. To the right, a detailed view of this meeting is shown. Another red circle highlights the "Join Zoom Meeting" section, which includes the URL <https://nyu.zoom.us/j/856462940> and the Meeting ID: 856 462 940. Below this, there are instructions on how to join by SIP and by H.323 with various IP addresses. At the bottom, it says "AMMI-NLP-2020 Created by: KyungHyun Cho".

Today < > March 2020

MON 16

GMT-04 AMMI Teaching

06:00

07:00 Lecture: Recap of neural ne
07:30, https://campuswire.c

08:00 Lecture: language modeling
08:00, https://campuswire.c

09:00

10:00 Lab: GCP and Jupyter Setup
09:30, https://campuswire.c

11:00 Lab: language modeling
11:00, https://campuswire.c

12:00

13:00 Lab: FF- PC Meet ICLR202
13:00, h 13:00 - 13:00, L

14:00

15:00

16:00

17:00

18:00 beer with keunwoo
18:00 - 20:15

19:00 Garage fMRI,

Lecture: language modeling
Monday, March 16 · 08:00 – 09:00

Join Zoom Meeting
<https://campuswire.com/c/GFADD516B/rooms/CF81B>

Kyunghyun Cho is inviting you to a scheduled Zoom meeting.

Join Zoom Meeting
<https://nyu.zoom.us/j/856462940>

Meeting ID: 856 462 940

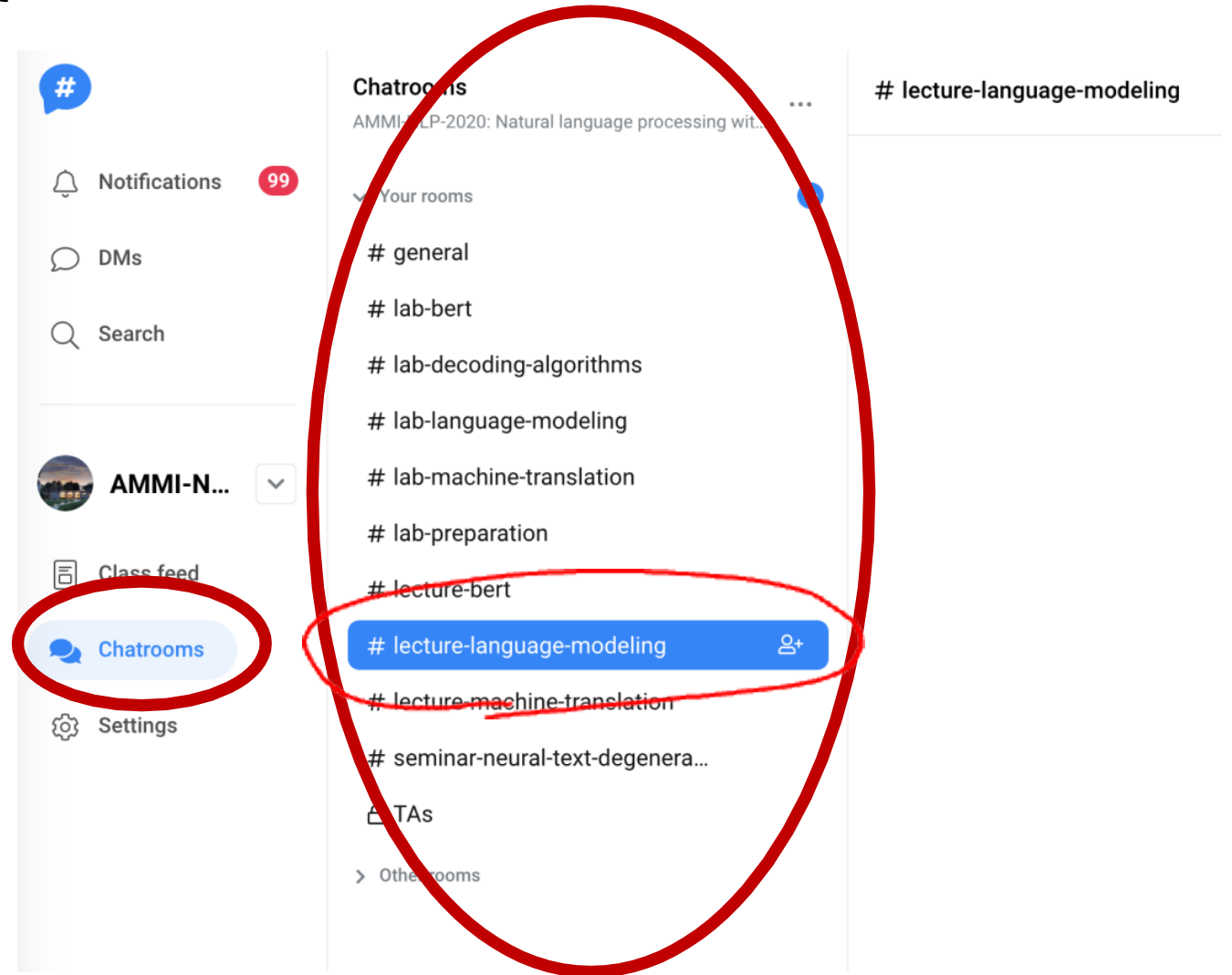
Join by SIP
856462940@zoomcrc.com

Join by H.323
162.255.37.11 (US West)
162.255.36.11 (US East)
221.122.88.195 (China)
115.114.131.7 (India Mumbai)
115.114.115.7 (India Hyderabad)
213.19.144.110 (EMEA)
103.122.166.55 (Australia)
209.9.211.110 (Hong Kong)
64.211.144.160 (Brazil)
69.174.57.160 (Canada)
207.226.132.110 (Japan)
Meeting ID: 856 462 940

AMMI-NLP-2020
Created by: KyungHyun Cho

Course discussion/Q&A

- URL:
<https://campuswire.com/p/GFADD516B>
 - Code: 9109
 - Please join now!
- Q&A using chatrooms



Course Assistants

- **Sreyas Mohan**
 - PhD student at NYU Center for Data Science
- **Phu Mon Htut**
 - PhD student at NYU Center for Data Science
- **Sean Welleck**
 - PhD student at NYU Computer Science



Recap: Supervised Machine Learning

You all know already, but it's important enough to warrant repetition.

Supervised Learning – Overview

- Provided:

1. a set of N input-output “training” examples

$$D = \{(x_1, y_1), \dots, (x_N, y_N)\}$$

2. A per-example loss function

$$l(M(x), y) \geq 0$$

3. Evaluation sets*: validation and test examples $D_{\text{val}}, D_{\text{test}}$

- What we must decide:

1. Hypothesis sets $\mathcal{H}_1, \dots, \mathcal{H}_M$

- Each set consists of all compatible models

2. Optimization algorithm

Often it is necessary to design a loss function.

* Often these sets are created by holding out subsets of training examples.

Supervised Learning – Overview

- Given:

1. $D = \{(x_1, y_1), \dots, (x_N, y_N)\}$ and $D_{\text{val}}, D_{\text{test}}$
2. $l(M(x), y) \geq 0$
3. $\mathcal{H}_1, \dots, \mathcal{H}_M$
4. Optimization algorithm

- Supervised learning finds an appropriate algorithm/model automatically

1. For each hypothesis set \mathcal{H}_m , find the best model:

$$\hat{M}_m = \arg \min_{M \in \mathcal{H}_m} \sum_{n=1}^N l(M(x_n), y_n)$$

using the optimization algorithm.

Supervised Learning – Overview

- Given:

1. $D = \{(x_1, y_1), \dots, (x_N, y_N)\}$ and $D_{\text{val}}, D_{\text{test}}$
2. $l(M(x), y) \geq 0$
3. $\mathcal{H}_1, \dots, \mathcal{H}_M$
4. Optimization algorithm

- Supervised learning finds an appropriate algorithm/model automatically

1. [Training] For each hypothesis set \mathcal{H}_m , find the best model:

$$\hat{M}_m = \arg \min_{\substack{M \in \mathcal{H}_m \\ \dots}} \sum_{n=1}^N l(M(x_n), y_n)$$

using the optimization algorithm and the **training set**.

Supervised Learning – Overview

- Given:

1. $D = \{(x_1, y_1), \dots, (x_N, y_N)\}$ and $D_{\text{val}}, D_{\text{test}}$
2. $l(M(x), y) \geq 0$
3. $\mathcal{H}_1, \dots, \mathcal{H}_M$
4. Optimization algorithm

- Supervised learning finds an appropriate algorithm/model automatically

2. [Model Selection]* Among the trained models, select the best one

$$\hat{M} = \arg \min_{M \in \{\mathcal{H}_1, \dots, \mathcal{H}_M\}} \sum_{(x, y) \in D_{\text{val}}} l(M(x), y)$$

using the validation set loss.

* If you're familiar with deep learning, “hyperparameter optimization” may be a more familiar term for you.

Supervised Learning – Overview

- Given:

1. $D = \{(x_1, y_1), \dots, (x_N, y_N)\}$ and $D_{\text{val}}, D_{\text{test}}$
2. $l(M(x), y) \geq 0$
3. $\mathcal{H}_1, \dots, \mathcal{H}_M$
4. Optimization algorithm


- Supervised learning finds an appropriate algorithm/model automatically

3. [Reporting] Report how well the best model *would* work

$$\underset{\dots\dots\dots}{R(\hat{M})} \approx \frac{1}{|D_{\text{test}}|} \sum_{(x,y) \in D_{\text{test}}} \underset{\text{test set}}{l(\hat{M}(x), y)}$$

using the **test set** loss.

Supervised Learning – Overview

- Given:
 1. $D = \{(x_1, y_1), \dots, (x_N, y_N)\}$ and $D_{\text{val}}, D_{\text{test}}$
 2. $l(M(x), y) \geq 0$
 3. $\mathcal{H}_1, \dots, \mathcal{H}_M$
 4. Optimization algorithm
 - Supervised learning finds an appropriate algorithm/model automatically
 - It results in an algorithm \hat{M} with an expected performance of $R(\hat{M})$
- 

Supervised Learning

- Three points to consider both in research and in practice
 1. How do we decide/design a **hypothesis set**?
 2. How do we decide a **loss function**?
 3. How do we **optimize** the loss function?

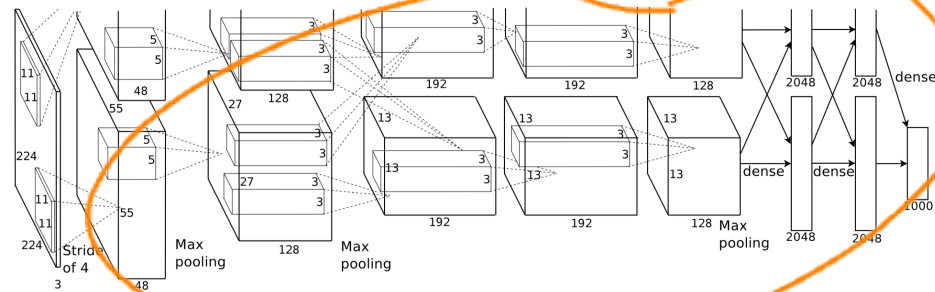
Hypothesis set – Neural Networks

- What kind of machine learning approach will we consider?
 - Classification:
 - Support vector machines, Naïve Bayes classifier, logistic regression, ...?
 - Regression:
 - Support vector regression, Linear regression, Gaussian process, ...?
- How are the hyperparameters sets?
 - Support vector machines: regularization coefficient C
 - Gaussian process: kernel function $k(\cdot, \cdot)$

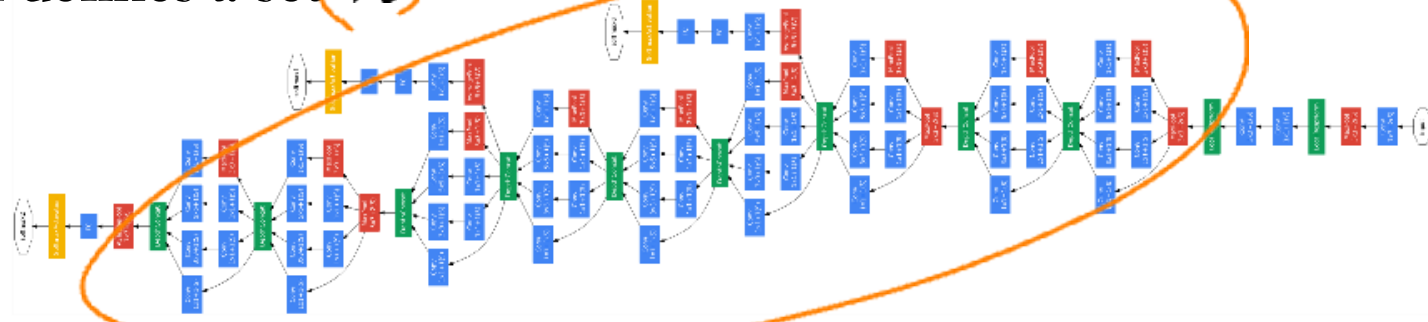
Hypothesis set – Neural Networks

- In the case of deep learning/artificial neural networks,

1. The architecture of a network defines a set \mathcal{H}



vs.

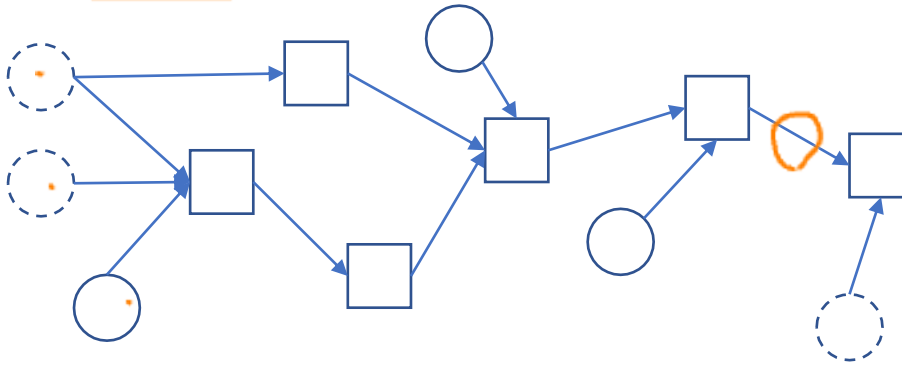





2. Each model in the set $M \in \mathcal{H}$ is characterized by its parameters θ .

- Weights and bias vectors define one model in the hypothesis set.
- There are infinitely many models in a hypothesis set.
- We use optimization to find “a” good model from the hypothesis set.

Network Architectures

- What is a neural network? – An (arbitrary) directed acyclic graph (DAG)



1. Solid Circles  : parameters (to be estimated or found)
2. Dashed Circles  : vector inputs/outputs (given as a training example)
3. Squares  : compute nodes (functions, often continuous/differentiable)

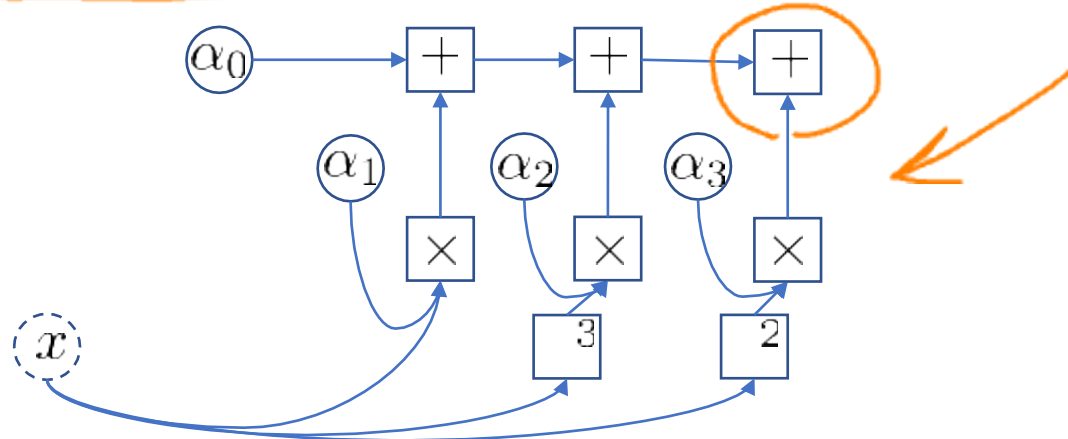
Network Architectures

- What is a neural network? – An (arbitrary) directed acyclic graph (DAG)

1. Logistic regression

$$p_{\theta}(y = 1|x) = \sigma(w^{\top}x + b) = \frac{1}{1 + \exp(-w^{\top}x - b)}$$

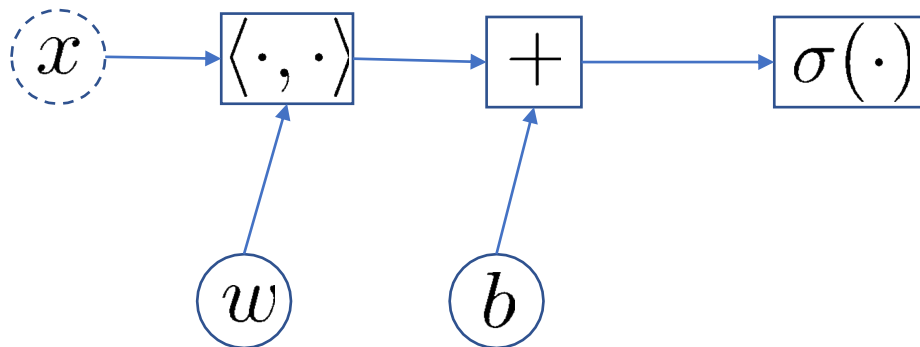
2. 3rd-order polynomial function $y = \alpha_0 + \alpha_1x + \alpha_2x^2 + \alpha_3x^3$



Inference – Forward Computation

- What is a neural network? – An (arbitrary) directed acyclic graph (DAG)
- Forward computation: how you “use” a trained neural network.
 - Topological sweep (breadth-first)
 - Logistic regression

$$p_{\theta}(y = 1|x) = \sigma(w^{\top}x + b) = \frac{1}{1 + \exp(-w^{\top}x - b)}$$



DAG \leftrightarrow Hypothesis Set

- What is a neural network? – An (arbitrary) directed acyclic graph (DAG)
- Implication in practice
 - Naturally supports high-level abstraction
 - Object-oriented paradigm fits well.*
 - Base classes: variable (input/output) node, operation node
 - Define the internal various types of variables and operations by inheritance
 - Maximal code reusability
 - See the success of PyTorch, TensorFlow, DyNet, ...
- You define a hypothesis set by designing a directed acyclic graph.
- The hypothesis space is then a set of all possible parameter settings.

↳ computation

Supervised Learning

- Three points to consider both in research and in practice
 1. How do we decide/design a **hypothesis set**?
 2. How do we decide a **loss function**?
 3. How do we **optimize** the loss function?

Loss Functions

- Per-example loss function
 - Computes how good a model is doing on a given example:
$$l(M(x), y) \geq 0$$
- So many loss functions...
 - Classification: hinge loss, log-loss, ...
 - Regression: mean squared error, mean absolute error, robust loss, ...
- In this lecture, we stick to distribution-based loss functions.

A Neural network

computes a conditional distribution

- Supervised learning: what is y given x ?

$$f_{\theta}(x) = ?$$

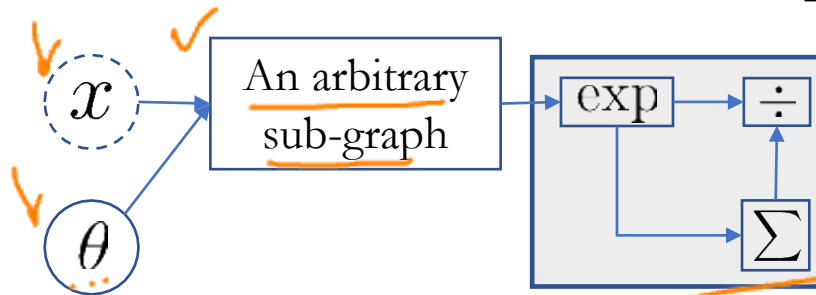
- In other words, how probable is a certain value y' of y given x ?

$$p(y = y' | x) = ?$$

- What kind of distributions?
 - Binary classification: Bernoulli distribution
 - Multiclass classification: Categorical distribution
 - Linear regression: Gaussian distribution
 - Multimodal linear regression: Mixture of Gaussians

Important distributions – Categorical

- How probable is a certain value y' of y given x ? $p(y = y' | x) = ?$
- Multi-class classification: Categorical distribution $\mathcal{C}(\{\mu_1, \mu_2, \dots, \mu_C\})$
 - Probability: $p(y = v | x) = \mu_v$, where $\sum_{v=1} \mu_v = 1$
 - Fully characterized by $\{\mu_1, \mu_2, \dots, \mu_C\}$.
 - A neural network then should turn the input x into a vector $\mu =$



$$\mu = \begin{bmatrix} \mu_1 \\ \mu_2 \\ \vdots \\ \mu_C \end{bmatrix}$$

using a softmax function: $\text{softmax}(a) = \frac{1}{\sum_{v=1}^C \exp(a_v)} \exp(a)$.

Important distributions – Gaussian

- How probable is a certain value y' of y given x ? $p(y = y'|x) = ?$
- Regression: Gaussian distribution $\mathcal{N}(\mu, \mathbb{I})$ with an identity covariance
 - Probability: $p(y|x) = \frac{1}{Z} \exp(-\frac{1}{2}(y - \mu)^\top (y - \mu))$
 - Fully characterized by $\mu \in \mathbb{R}^q$.
 - A neural network then should turn the input x into a vector μ .
 - Can be done trivially by affine transformation.

Loss Function – negative log-probability

- Once a neural network outputs a conditional distribution $p_\theta(y|x)$, a natural way to define a loss function arises.
- Make sure training data is maximally likely:
 - Equiv. to making sure each and every training example is maximally likely.

$$\arg \max_{\theta} \log p_{\theta}(D) = \arg \max_{\theta} \sum_{n=1}^N \log p_{\theta}(y_n|x_n) \quad \text{correct}$$

- Why \log ? – many reasons... but out of the lecture's scope.
- Equivalently, we want to minimize the negative log-probability.
 - A loss function is the sum of negative log-probabilities of correct answers.

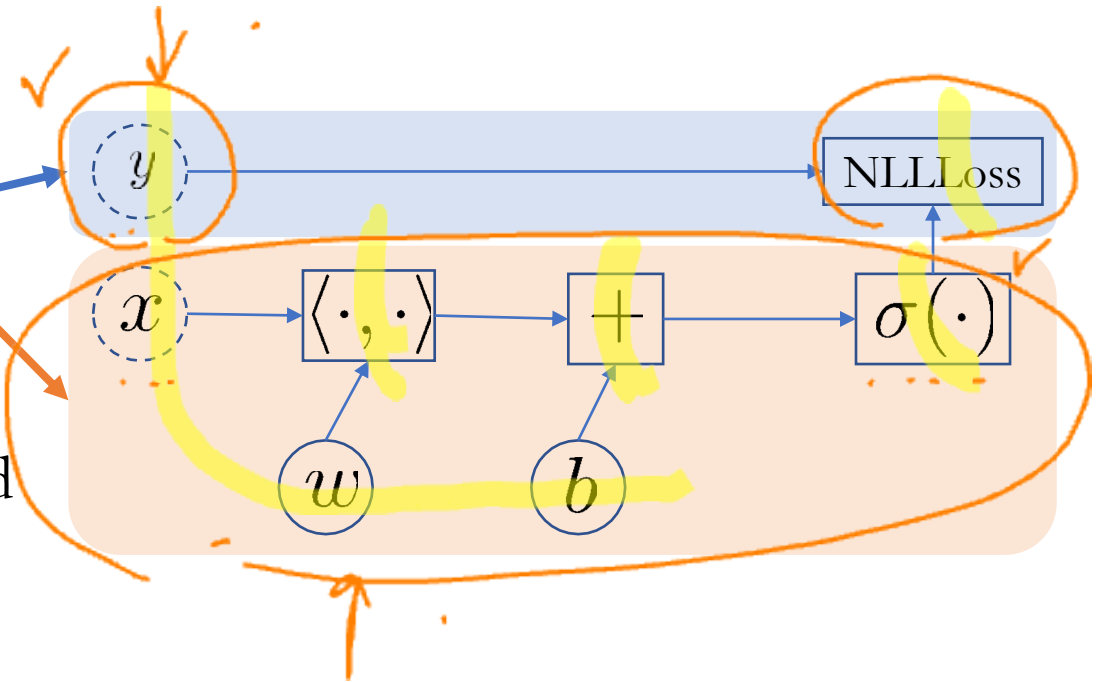
$$L(\theta) = \sum_{n=1}^N l(M_{\theta}(x_n), y_n) = - \sum_{n=1}^N \log p_{\theta}(y_n|x_n)$$

Loss Function – negative log-probability

- Once a neural network outputs a conditional distribution $p_{\theta}(y|x)$, a natural way to define a loss function arises.
- Practical implications
 - An OP node: negative log-probability (e.g., NLLLoss in PyTorch)
 - Inputs: the conditional distribution and the correct output
 - Output: the negative log-probability (a scalar)

Loss Function – negative log-probability

- Once a neural network outputs a conditional distribution $p_{\theta}(y|x)$, a natural way to define a loss function arises.
- Logistic regression
 - Computes a Bernoulli distribution
 - Computes a negative log-probability
 - All in **one directed acyclic graph**
- Forward computation
 - Computes the conditional distribution, and
 - Computes the per-example loss



Supervised Learning

- Three points to consider both in research and in practice
 1. How do we decide/design a **hypothesis set**?
 2. How do we decide a **loss function**?
 3. How do we **optimize** the loss function?

Loss Minimization

- What we now know
 1. How to build a neural network with an arbitrary architecture.
 2. How to define a per-example loss as a negative log-probability.
 3. Define a single directed acyclic graph containing both.
- What we now need to know
 1. Choose an optimization algorithm.
 2. How to use the optimization algorithm to estimate parameters θ .

Gradient-based optimization

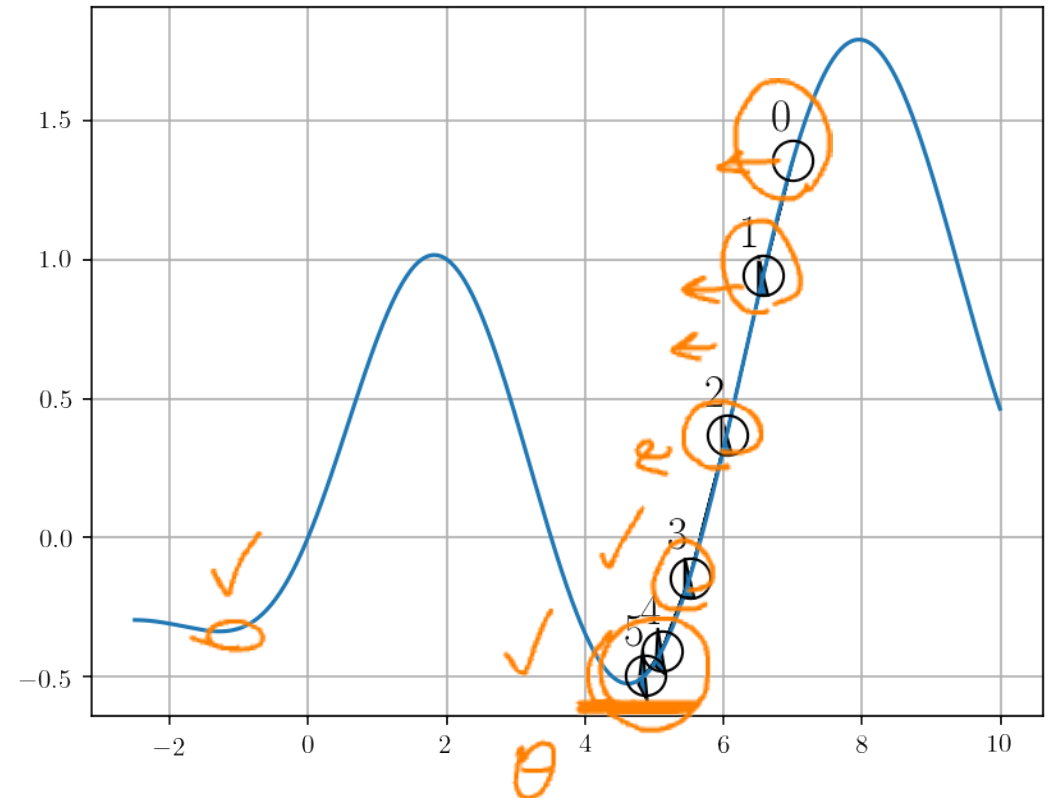
- A continuous, differentiable* function $L: \mathbb{R}^d \rightarrow \mathbb{R}$
- Given the current value θ_0 , how should I move to minimize L ?
- Gradient descent
 - The negative gradient of the function: $-\nabla L(\theta_0)$
 - This is only valid in a local neighbourhood of θ_0 take a very small step!
$$\theta = \theta_0 - \eta \nabla L(\theta_0)$$
- Efficient and effective even in the high dimensional space.
 - Can be improved with the second-order information (Hessian and/or FIM)

* Almost everywhere, but not necessarily everywhere ²⁸

Gradient-based optimization

- A **continuous, differentiable** function $L : \mathbb{R}^d \rightarrow \mathbb{R}$
- Given the current value θ_0 , how should I move to minimize L ?
- Gradient descent
 - Efficient and effective even in the high dimensional space.
 - Learning rate must be carefully selected and annealed over time.

2



Backward Computation – Backpropagation

- How do we compute the gradient of the loss function?

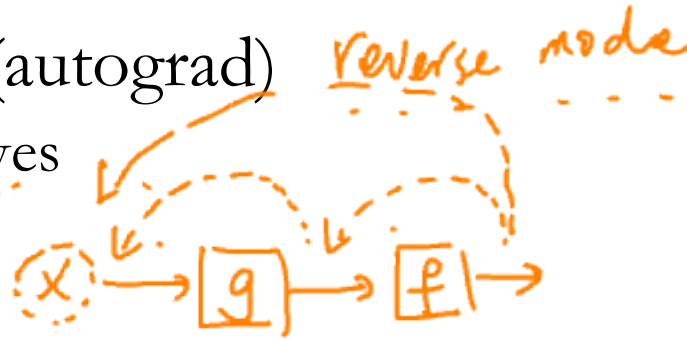
1. Manual derivation

- Relatively doable when the DAG is small and simple.
- When the DAG is larger and complicated, too much hassle.

2. Automatic differentiation (autograd)

- Use the chain rule of derivatives

$$\frac{\partial (f \circ g)}{\partial x} = \frac{\partial f}{\partial g} \frac{\partial g}{\partial x}$$



- The DAG is nothing but a composition of (mostly) differentiable functions.
- Automatically apply the chain rule of derivatives.

Backward Computation – Backpropagation

- Automatic differentiation (autograd)

1. Implement the Jacobian-vector product of each OP node:

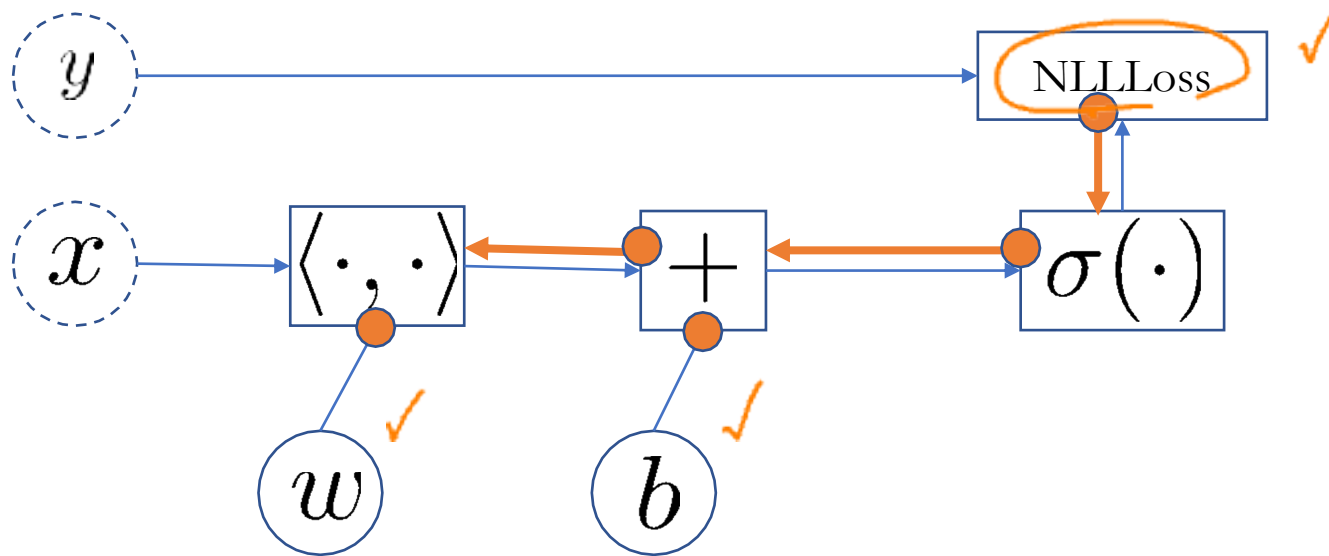
$$\begin{bmatrix} \frac{\partial L}{\partial x_1} \\ \vdots \\ \frac{\partial L}{\partial x_d} \end{bmatrix} = \begin{bmatrix} \frac{\partial F_1}{\partial x_1} & \dots & \frac{\partial F_{d'}}{\partial x_1} \\ \vdots & \ddots & \vdots \\ \frac{\partial F_1}{\partial x_d} & \dots & \frac{\partial F_{d'}}{\partial x_d} \end{bmatrix} \begin{bmatrix} \frac{\partial L}{\partial F_1} \\ \vdots \\ \frac{\partial L}{\partial F_{d'}} \end{bmatrix}$$

Handwritten notes: *func*, $F: \mathbb{R}^d \rightarrow \mathbb{R}^{d'}$, *Jacobian F*.

- Can be implemented efficiently without explicitly computing the Jacobian.
- The same implementation can be reused every time the OP node is called.

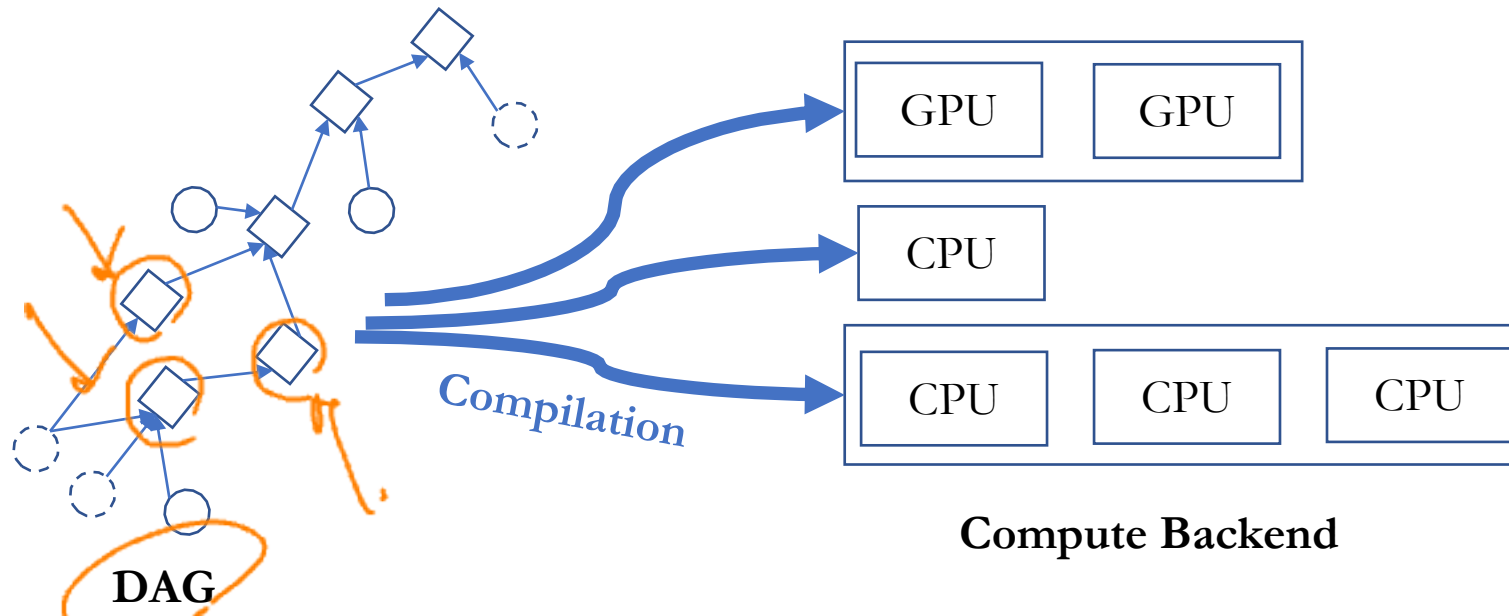
Backward Computation – Backpropagation

- Automatic differentiation (autograd)
 2. Reverse-sweep the DAG starting from the loss function node.
 - Iteratively multiplies the Jacobian of each OP node until the leaf nodes of the parameters.
 - As expensive as forward computation with a constant overhead: $O(N)$, where N : # of nodes.



Backward Computation – Backpropagation

- Practical Implications – Automatic differentiation (autograd)
 - Unless a complete new OP is introduced, no need to manually derive the gradient
 - Nice de-coupling of specification (front-end) and implementation (back-end)
 1. [Front-end] Design a neural network by creating a DAG.
 2. [Back-end] The DAG is “compiled” into an efficient code for a target compute device.



Gradient-based Optimization

- Backpropagation gives us the gradient of the loss function w.r.t. θ
- Readily used by off-the-shelf gradient-based optimizers
 - Gradient descent, L-BFGS, Conjugate gradient, ...
 - Though, most are not applicable in a realistic neural network with 10s or 100s of millions of parameters.
- **Stochastic gradient descent**
 - Approximate the full loss function (the sum of per-examples losses) using only a small random subset of training examples:

$$\nabla L \approx \frac{1}{N'} \sum_{n=1}^{N'} \nabla l(M(x_{n'}), y_{n'})$$

*mini batch
gradient*

Stochastic Gradient Descent

- Stochastic gradient descent
 - Approximate the full loss function (the sum of per-examples losses) using only a small random subset of training examples:

$$\nabla L \approx \frac{1}{N'} \sum_{n=1}^{N'} \nabla l(M(x_{n'}), y_{n'})$$

- Unbiased estimate of the full gradient.*
- Learning rate must be annealed appropriately.
- Extremely efficient *de facto* standard practice.

...

* Under certain conditions

Stochastic Gradient Descent

- Stochastic gradient descent in practice

1. Grab a random subset of M training examples*

$$D' = \{(x_1, y_1), \dots, (x_{N'}, y_{N'})\}$$

2. Compute the minibatch gradient

$$\nabla L \approx \frac{1}{N'} \sum_{n=1}^{N'} \nabla l(M(x_{n'}), y_{n'})$$

3. Update the parameters

$$\theta \leftarrow \theta + \eta \nabla L(\theta; D')$$

4. Repeat until the validation loss stops improving.

* In practice, sample without replacement until the training set is exhausted (one epoch).

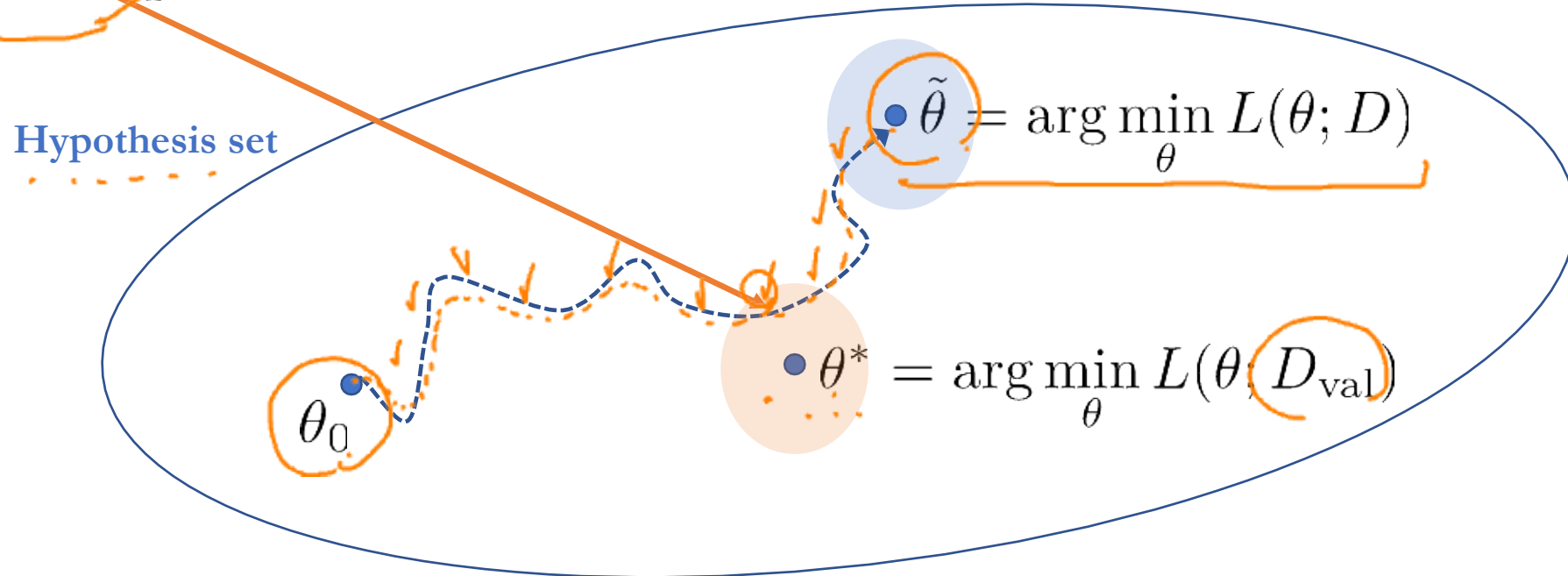
This is called early-stopping which prevents the neural network from overfitting to training examples.

Stochastic Gradient Descent – Early Stopping

- Stochastic gradient descent in practice
 1. Grab a random subset of M training examples
 2. Compute the minibatch gradient
 3. Update the parameters
 4. **Repeat until the validation loss stops improving.**
- An efficient way to prevent overfitting
 - Overfitting: the training loss is low, but the validation loss is not.
 - The most serious problem in statistical machine learning.
 - Early-stop based on the validation loss

Stochastic Gradient Descent – Early Stopping

- An efficient way to prevent overfitting
 - Overfitting: the training loss is low, but the validation loss is not.
 - The most serious problem in statistical machine learning.
 - **Early-stop** based on the validation loss



Stochastic Gradient Descent

– Adaptive Learning Rate

- Stochastic gradient descent in practice
 1. Grab a random subset of M training examples $D' = \{(x_1, y_1), \dots, (x_{N'}, y_{N'})\}$
 2. Compute the minibatch gradient
 3. Update the per-parameter learning rate η_θ
 4. Update the parameters
$$\theta \leftarrow \theta - \eta_\theta \frac{\partial L'}{\partial \theta}$$
 5. Repeat until the validation loss stops improving.
- Adaptive learning rate: Adam [Kingma&Ba, 2015], Adadelata [Zeiler, 2015], and many more...
 - Approximately re-scale parameters to improve the conditioning of the Hessian.

Supervised Learning with Neural Networks

1. How do we decide/design a **hypothesis set**?
 - Design a network architecture as a directed acyclic graph
2. How do we decide a **loss function**?
 - Frame the problem as a conditional distribution modelling
 - The per-example loss function is a negative log-probability of a correct answer
3. How do we **optimize** the loss function?
 - Automatic backpropagation: no manual gradient derivation
 - Stochastic gradient descent with early stopping [and adaptive learning rate]

Language modeling as supervised learning

On the boundary between unsupervised and supervised learning

Language Modelling

- Input: a sentence *paragraph ~*
- Output: the probability of the input sentence
- A language model captures the distribution over all possible sentences.

$$p(X) = p(\underbrace{x_1, x_2, \dots, x_T}_{\dots})$$

- It is unsupervised learning.
 - We will however turn the problem into a *sequence of supervised learning*.

Autoregressive language modelling

- Autoregressive sequence modelling
 - The distribution over the next token is based on all the previous tokens.
$$p(X) = p(x_1)p(x_2|x_1) \cdots p(x_T|x_1, \dots, x_{T-1})$$
 - This equality holds exactly due to the def. of conditional distribution.
- Unsupervised learning becomes a set of supervised problems.
 - Each conditional is a neural network classifier.
 - Input is all the previous tokens (a partial sentence).
 - Output is the distribution over all possible next tokens (classes).
 - It is a text classification problem.

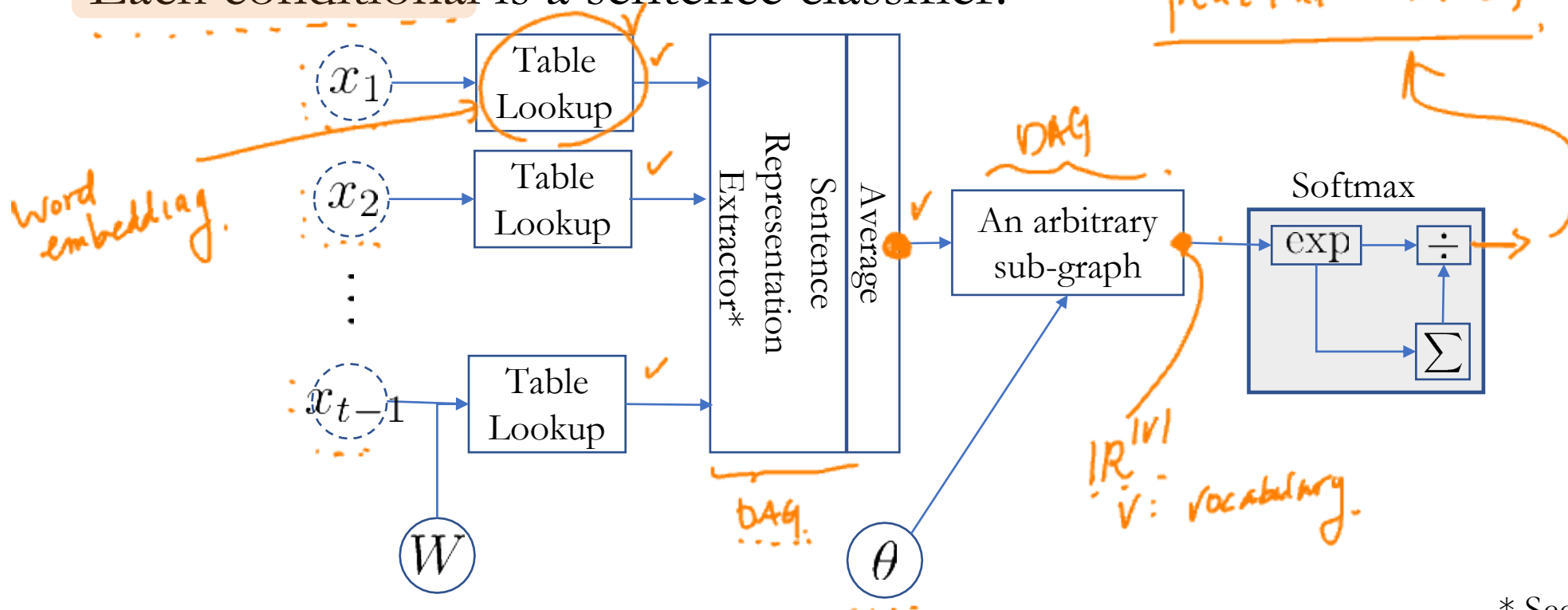
↑
vocabulary

Autoregressive language modelling

- Autoregressive sequence modelling
 - The distribution over the next token is based on all the previous tokens.

$$p(X) = p(x_1)p(x_2|x_1) \cdots p(x_T|x_1, \dots, x_{T-1})$$

- Each conditional is a sentence classifier:

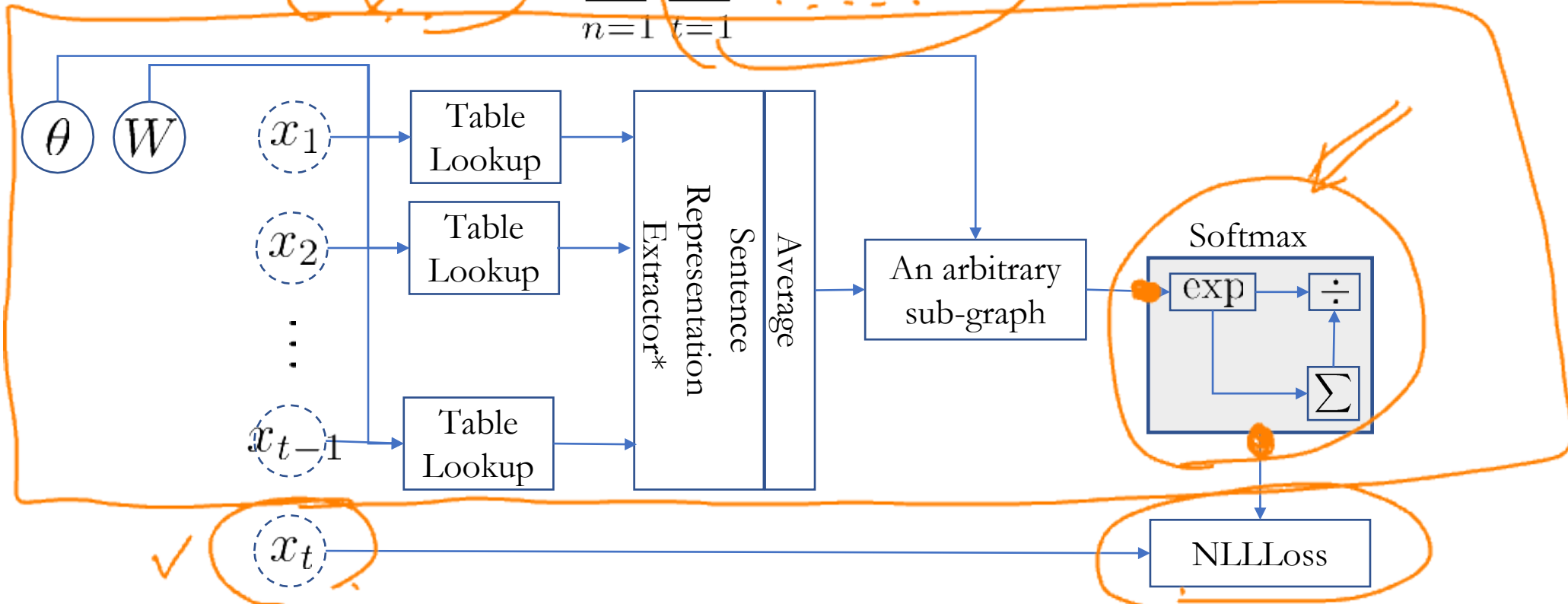


Autoregressive language modelling

- Autoregressive sequence modelling $p(X) = \prod_{t=1}^T p(x_t | x_{<t})$
- Loss function: the sum of negative log probabilities

$$\log p_{\theta}(X) = \sum_{n=1}^N \sum_{t=1}^T \log p_{\theta}(x_t | x_{<t})$$

self-supervised lang



Scoring a sentence

- Autoregressive sequence modelling
 - The distribution over the next token is based on all the previous tokens.

$$p(X) = p(x_1)p(x_2|\tilde{x}_1) \cdots p(x_T|\tilde{x}_1, \dots, \tilde{x}_{T-1}) \Leftrightarrow \log p(X) = \sum_{t=1}^T \log p(x_t | x_{<t})$$

- A natural way to score a sentence:
 - In Korea, more than half of residents speak Korean.
 - “In” is a reasonable token to start a sentence.
 - “Korea” is pretty likely given “In”
 - “more” is okay token to follow “In Korea”
 - “than” is very likely after “In Korea, more”
 - “half” is also very likely after “In Korea, more than”
 - ⋮
- Sum all these scores and get the sentence score.

Scoring a sentence

- Autoregressive sequence modelling
 - The distribution over the next token is based on all the previous tokens.
$$p(X) = p(x_1)p(x_2|x_1) \cdots p(x_T|x_1, \dots, x_{T-1})$$
- A natural way to score a sentence:
 - “In Korea, more than half of residents speak Korean.”
vs.
“In Korea, more than half of residents speak Finnish.”
 - The former is more likely (=higher probability) than the latter.
- This is precisely what NLLLoss computes over the sentence.

N-Gram Language Models

- Let's back up a little...
- What would we do without a neural network?
- Assume a Markovian property.

$$p(X) = \prod_{t=1}^T p(x_t | x_{<t}) \approx \prod_{t=1}^T p(x_t | \underbrace{x_{t-n}, \dots, x_{t-1}}_{n < \infty})$$

limit.

$x_1 \dots x_{t-1}$	x_t	$p(x_t x_1 \dots x_{t-1})$
_____	_____	_____
_____	_____	_____
_____	_____	_____
_____	_____	_____
_____	_____	_____

probability table.

- This turned out to be crucial, and we will discuss why shortly.

N-Gram Language Models

$$p(X) = \prod_{t=1}^T p(x_t | x_{<t}) \approx \prod_{t=1}^T p(x_t | x_{t-n}, \dots, x_{t-1})$$

- We need to estimate n-gram probabilities: $p(\underbrace{x}_{\text{...}} | x_{-N}, x_{-N+1}, \dots, x_{-1})$
- Recall the def. of conditional and marginal probabilities:

$$\begin{aligned} \underline{p(x | x_{-N}, x_{-N+1}, \dots, x_{-1})} &= \frac{p(\underline{x_{-N}, x_{-N+1}, \dots, x_{-1}, x})}{p(\underline{x_{-N}, x_{-N+1}, \dots, x_{-1}})} \\ &= \frac{p(x_{-N}, x_{-N+1}, \dots, x_{-1}, x)}{\sum_{\underline{x \in V}} p(x_{-N}, x_{-N+1}, \dots, x_{-1}, \underline{x})} \end{aligned}$$

Handwritten notes:
- An arrow points from "n-gram" to the numerator of the first equation.
- "cond. prob." is written next to the first equation.
- "marginalization" is written with an arrow pointing to the summation in the second equation.

- V : all possible tokens (=vocabulary)

N-Gram Language Models

- We need to estimate n -gram probabilities:

$$p(x|x_{-N}, x_{-N+1}, \dots, x_{-1}) = \frac{p(x_{-N}, x_{-N+1}, \dots, x_{-1}, x)}{\sum_{x \in V} p(x_{-N}, x_{-N+1}, \dots, x_{-1}, x)}$$

- How do we estimate the probability?
 - I want to estimate the probability of my distorted coin landing head.
 - **Maximum likelihood estimation (MLE)**:
toss the coin a lot and look at how often it lands heads.

Data Collection

Estimation

N-Gram Language Models

- We need to estimate n -gram probabilities:

$$p(x|x_{-N}, x_{-N+1}, \dots, x_{-1}) = \frac{p(x_{-N}, x_{-N+1}, \dots, x_{-1}, x)}{p(x_{-N}, x_{-N+1}, \dots, x_{-1})}$$

- Data: all the documents or sentences you can collect

- e.g., Wikipedia, news articles, tweets, ...

- Estimation:

1. Count the # of occurrences for the n -gram $(x_{-N}, x_{-N+1}, \dots, x_{-1}, x)$
2. Count the #'s of occurrences for all the n -grams of the form:

$$(x_{-N}, x_{-N+1}, \dots, x_{-1}, ?)$$

N-Gram Language Models

- We need to estimate n -gram probabilities:

$$p(x|x_{-N}, x_{-N+1}, \dots, x_{-1}) = \frac{p(x_{-N}, x_{-N+1}, \dots, x_{-1}, x)}{p(x_{-N}, x_{-N+1}, \dots, x_{-1})}$$

- Estimation:

$$\hat{p}(x|x_{-N}, x_{-N+1}, \dots, x_{-1}) = \frac{p(x_{-N}, x_{-N+1}, \dots, x_{-1}, x)}{\sum_{x \in V} p(x_{-N}, x_{-N+1}, \dots, x_{-1}, x)}$$
$$\approx \frac{c(x_{-N}, x_{-N+1}, \dots, x_{-1}, x)}{\sum_{x' \in V} c(x_{-N}, x_{-N+1}, \dots, x_{-1}, x')}$$

count

- *Do you see why this makes sense?*

N-Gram Language Models

- We need to estimate n-gram probabilities:

$$p(x|x_{-N}, x_{-N+1}, \dots, x_{-1}) = \frac{p(x_{-N}, x_{-N+1}, \dots, x_{-1}, x)}{\sum_{x \in V} p(x_{-N}, x_{-N+1}, \dots, x_{-1}, x)}$$

$\approx \frac{c(x_{-N}, x_{-N+1}, \dots, x_{-1}, x)}{\sum_{x' \in V} c(x_{-N}, x_{-N+1}, \dots, x_{-1}, x')}$

Handwritten notes: A checkmark is next to the approximation symbol. The numerator and denominator of the approximation are boxed. Arrows point from the boxes to the right.

- How likely is “University” given “New York”? *Handwritten: $p(\text{University} | \text{New, York})$*
 - Count all “New York University”
 - Count all “New York ?”: e.g., “New York State”, “New York City”, “New York Fire”, “New York Police”, “New York Bridges”, ...
 - How often “New York University” happens among these?

N-Gram Language Models – Two problems

1. Data sparsity: lack of generalization

- What happens “one” n-gram never happens?

$$\begin{aligned} p(\text{a lion is chasing a llama}) &= p(a) \times p(\text{lion}|a) \times p(\text{is}|a \text{ lion}) \\ &\times p(\text{chasing}|\text{lion is}) \times p(a|\text{is chasing}) \\ &\times \underbrace{p(\text{llama}|\text{chasing a})}_{=0} = 0 \end{aligned}$$

Handwritten notes: $|V|^N$ and a checkmark above the llama illustration.

2. Inability to capture long-term dependencies

- Each conditional only considers a small window of size n .
- Consider *“the same **stump** which had impaled the car of many a guest in the past thirty years and which **he refused to have removed**”*
- It is impossible to tell “removed” is likely by looking at the four preceding tokens.

Traditional Solutions

1. Data Sparsity

- Smoothing: add a small constant to avoid 0.

$$p(x|x_{-N}, x_{-N+1}, \dots, x_{-1}) \approx \frac{c(x_{-N}, x_{-N+1}, \dots, x_{-1}, x) + \epsilon}{\epsilon|V| + \sum_{x' \in V} c(x_{-N}, x_{-N+1}, \dots, x_{-1}, x')}$$

- Backoff: try a shorter window.

$$c(x_{-N}, \dots, x) = \begin{cases} \alpha c(x_{-N+1}, \dots, x) + \beta, & \text{if } c(x_{-N}, \dots, x) = 0 \\ c(x_{-N}, \dots, x), & \text{otherwise} \end{cases}$$

parameters
↓

- The most widely used approach: Kneser-Ney smoothing/backoff
- KenLM implements the efficient n-gram LM model.

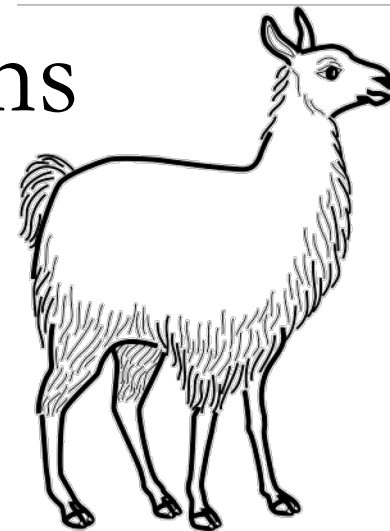
Traditional Solutions

2. Long-Term Dependency

- Increase n : not feasible as the data sparsity worsens.
 - # of all possible n -grams grows exponentially w.r.t. n : $O(|V|^n)$
 - The data size does not grow exponentially: many never-occurring n -grams.
-
- These two problems are closely related and cannot be tackled well.
 - To capture long-term dependencies, n must be large.
 - To address data sparsity, n must be small.
 - Conflicting goals..

Count-based

N-Gram Language Models – Two problems



1. Data sparsity: lack of generalization

- What happens “one” n-gram never happens?

$$\begin{aligned} p(\text{a lion is chasing a llama}) &= p(\text{a}) \times p(\text{lion}|\text{a}) \times p(\text{is}|\text{a lion}) \\ &\quad \times p(\text{chasing}|\text{lion is}) \times p(\text{a}|\text{is chasing}) \\ &\quad \times \underbrace{p(\text{llama}|\text{chasing a})}_{=0} = 0 \end{aligned}$$

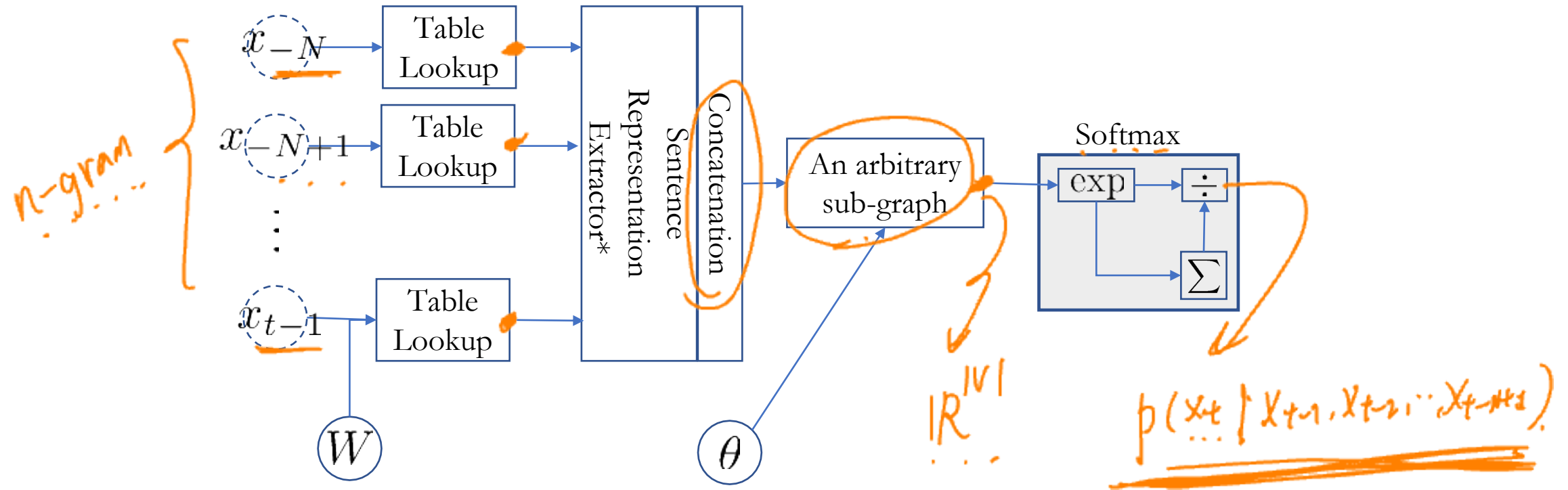
2. Inability to capture long-term dependencies

- Each conditional only considers a small window of size n .
- Consider “*the same **stump** which had impaled the car of many a guest in the past thirty years and which **he refused to have removed***”
- It is impossible to tell “removed” is likely by looking at the four preceding tokens.

Feed forward

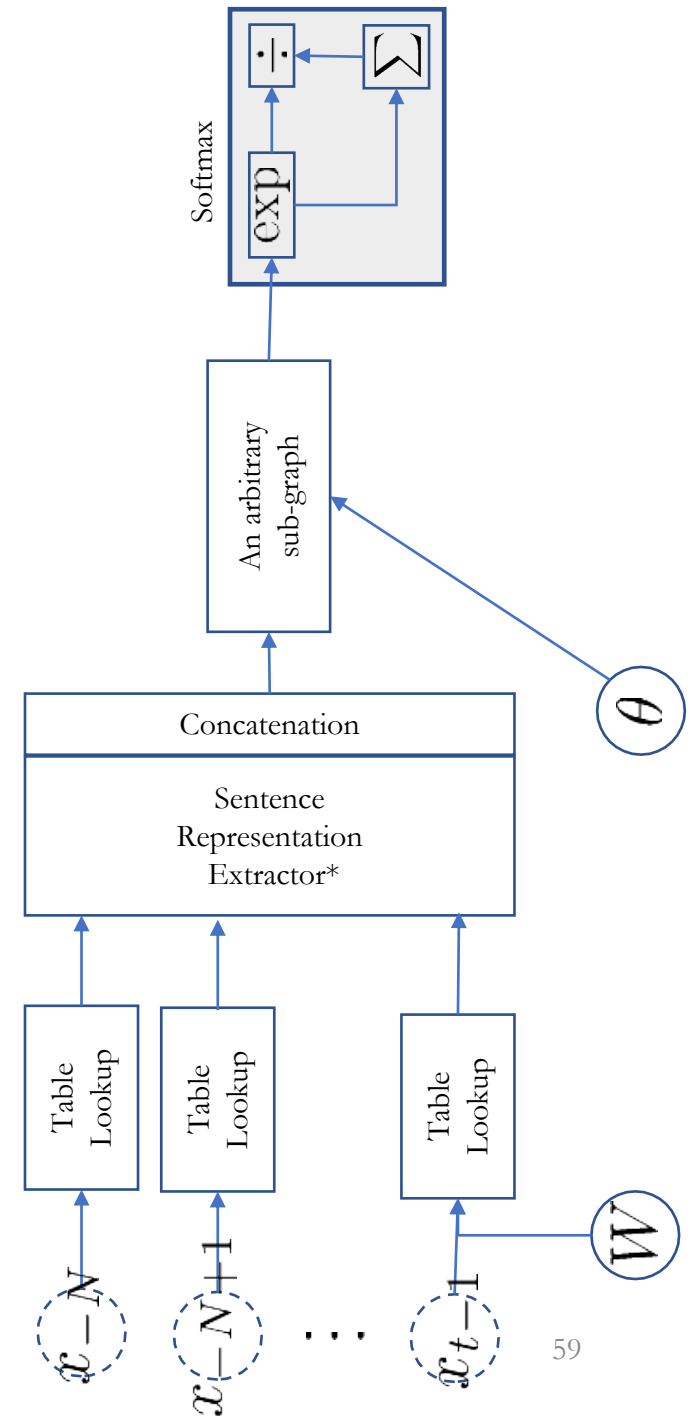
Neural N-Gram Language Model [Bengio et al., 2001]

- The first extension of n-gram language models using a neural network



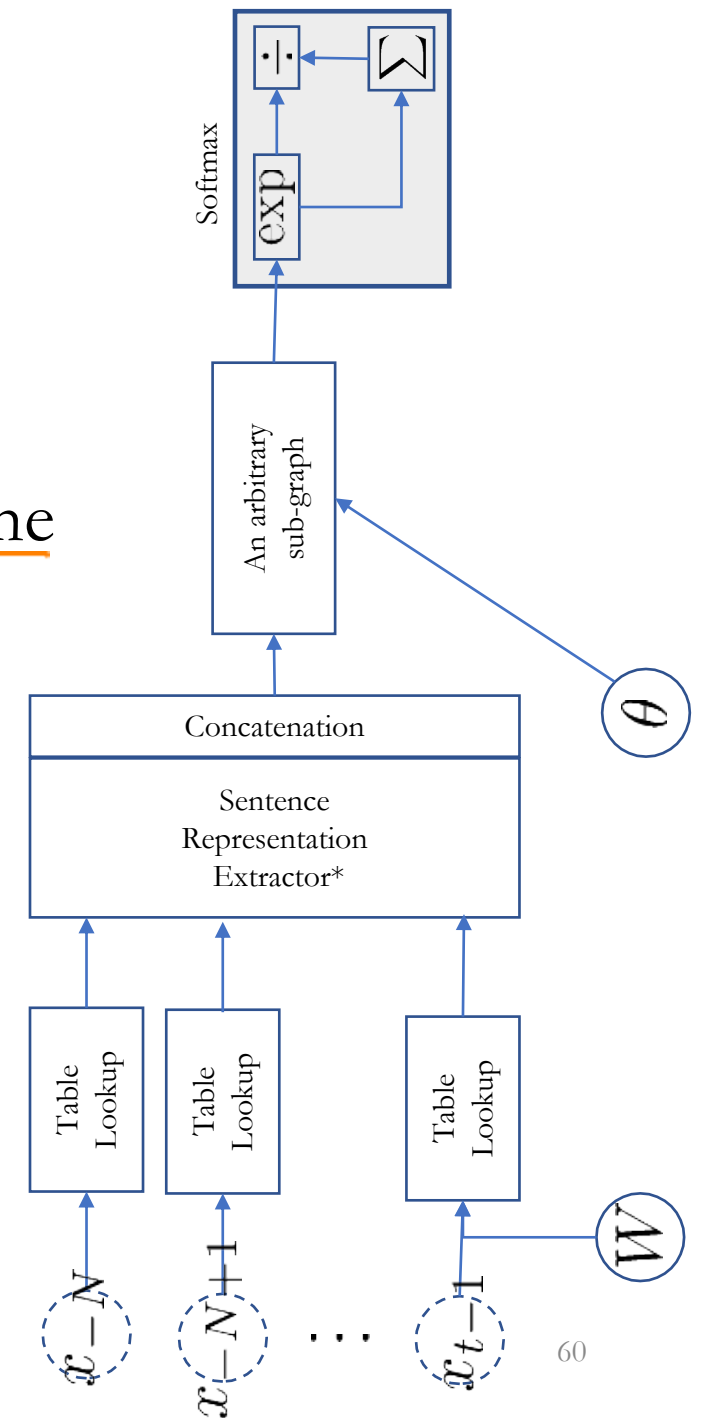
Neural N-Gram Language Model

- The first neural language models
- Trained using backpropagation and SGD
- Generalizes to an unseen n -gram
- Addresses the issue of data sparsity
- *How?*



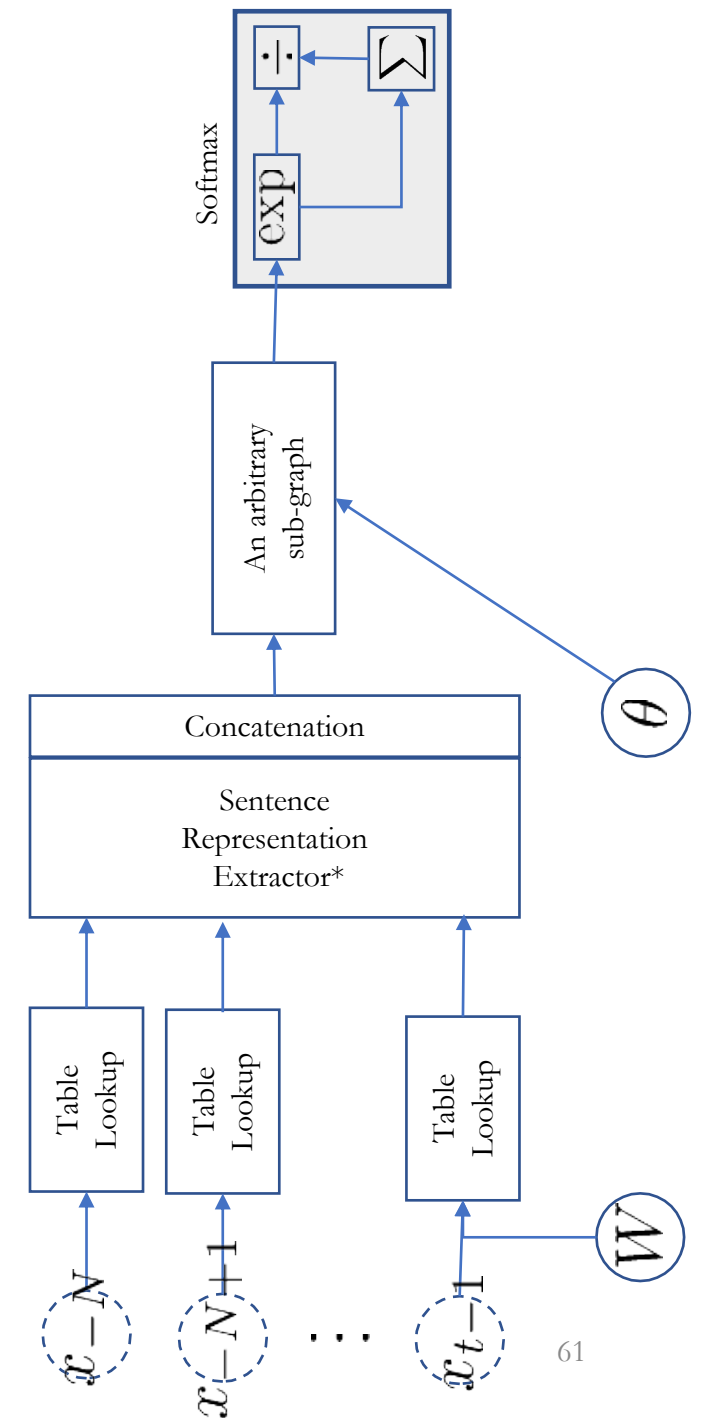
Neural N-Gram Language Model

- Why does the data sparsity happen?
- A “shallow” answer: some n-grams do not occur in the training data, while they do in the test time.
- A “slightly deeper” answer: it is difficult to impose token/phrase similarities in the discrete space.



Neural N-Gram Language Model

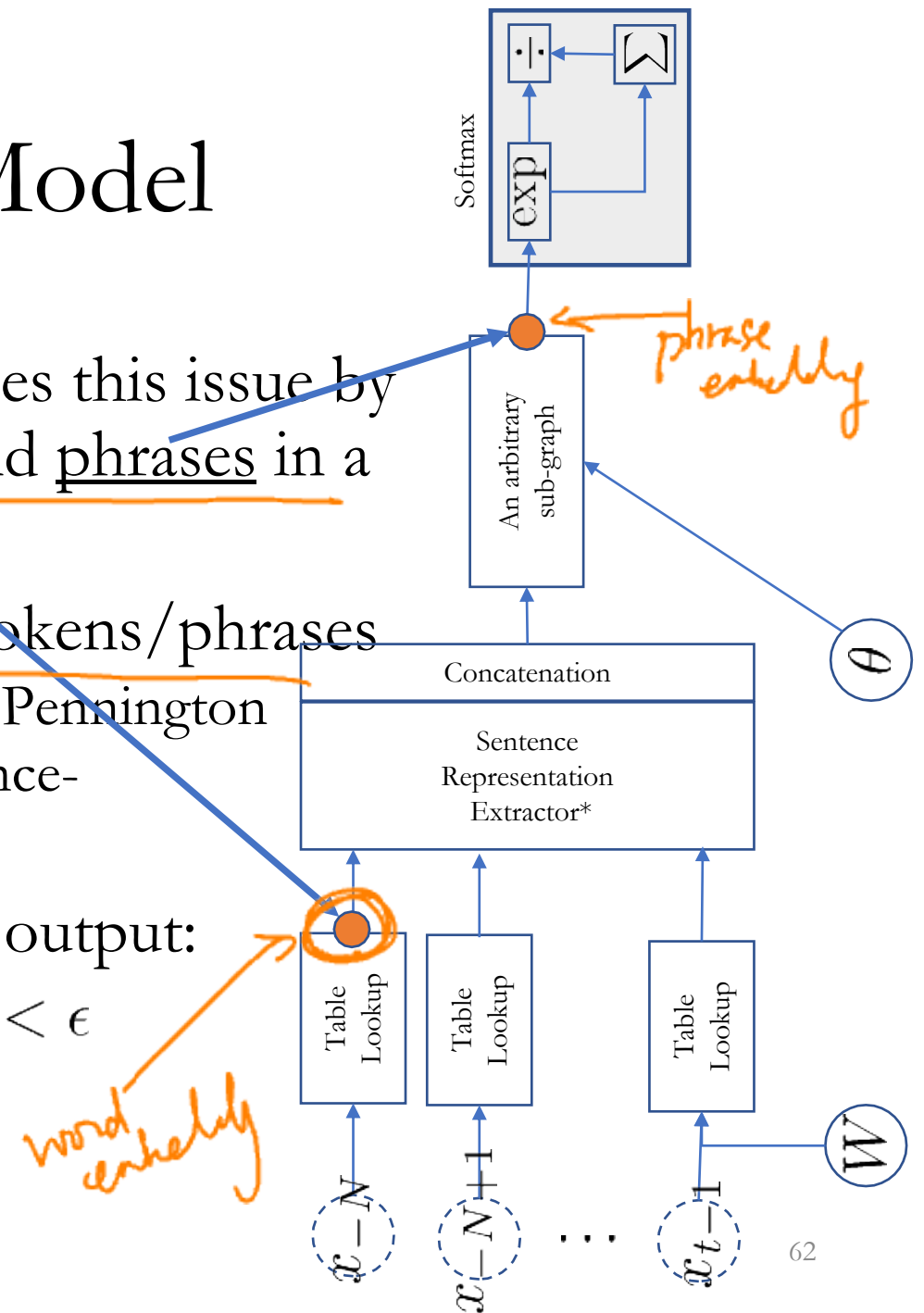
- Why does the data sparsity happen?
- Back to the earlier example
 - Problem: $c(\text{chasing a llama}) = 0$ ✗
 - Observation: $c(\text{chasing a cat}) \gg 0$
 $c(\text{chasing a dog}) \gg 0$
 $c(\text{chasing a deer}) \gg 0$
- If the LM knew “llama” is a mammal similar to “cat”, “dog” and “deer”, it would be able to guess “chasing a llama” is as likely as “chasing a cat”, “chasing a dog”, and “chasing a deer”.



Neural N-Gram Language Model

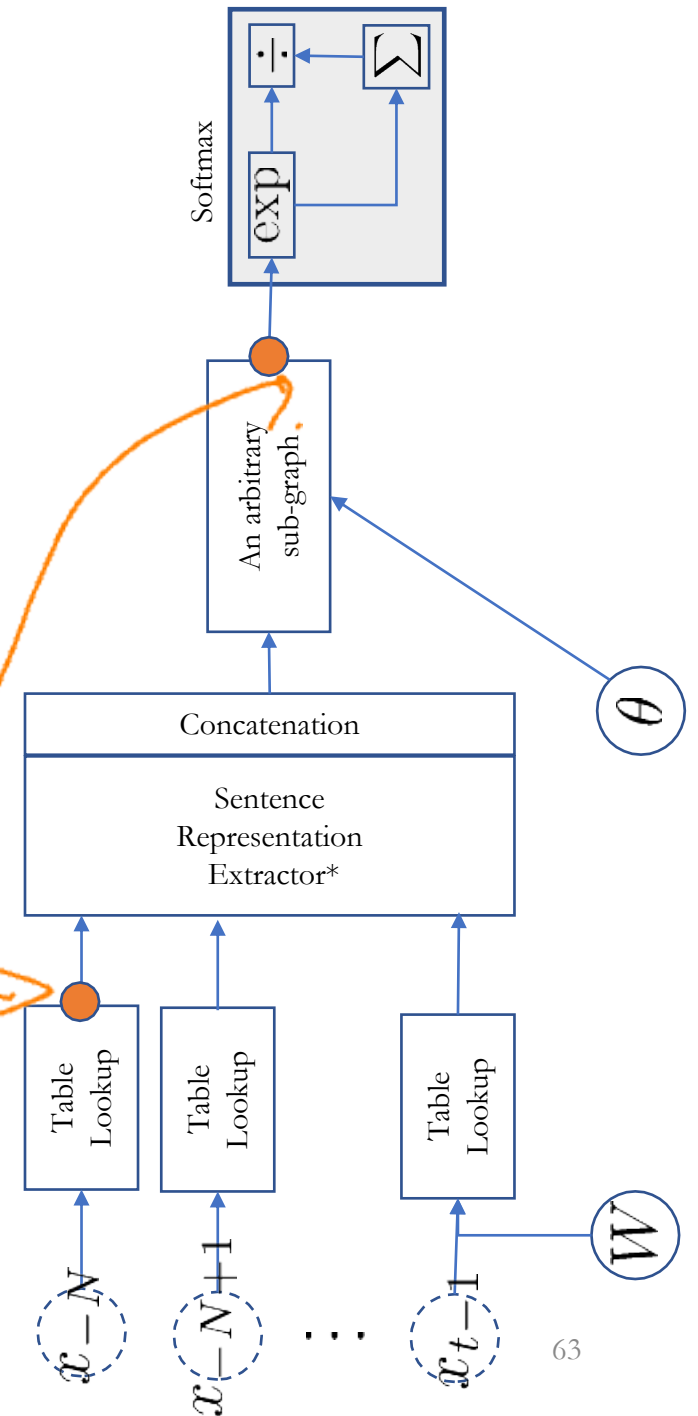
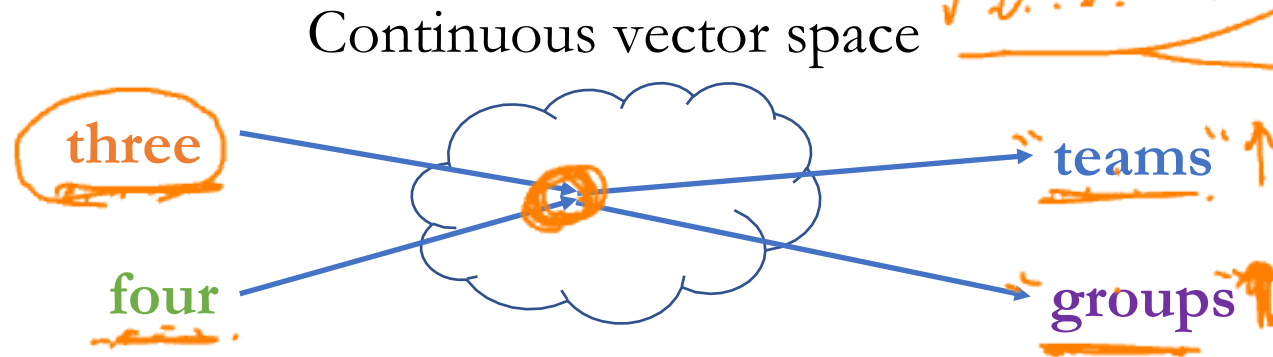
- The neural n-gram language model addresses this issue by “learning the similarities” among tokens and phrases in a “continuous vector space”.
- In the “continuous vector space”, similar tokens/phrases are nearby: e.g., word2vec [Mikolov et al., 2013; Pennington et al., 2014], doc2vec [Le&Mikolove, 2014], sentence-to-vec [Hill et al., 2016 and ref’s therein]
- Then, similar input n-grams lead to similar output:

$$D(x_t | x_{t-N}, \dots, x_{t-1} || x_t | x'_{t-N}, \dots, x'_{t-1}) < \epsilon$$



Neural N-Gram Language Model

- Training examples
 - there are three teams left for qualification.
 - four teams have passed the first round.
 - four groups are playing in the field.
- Q: how likely is “groups” followed by “three”?



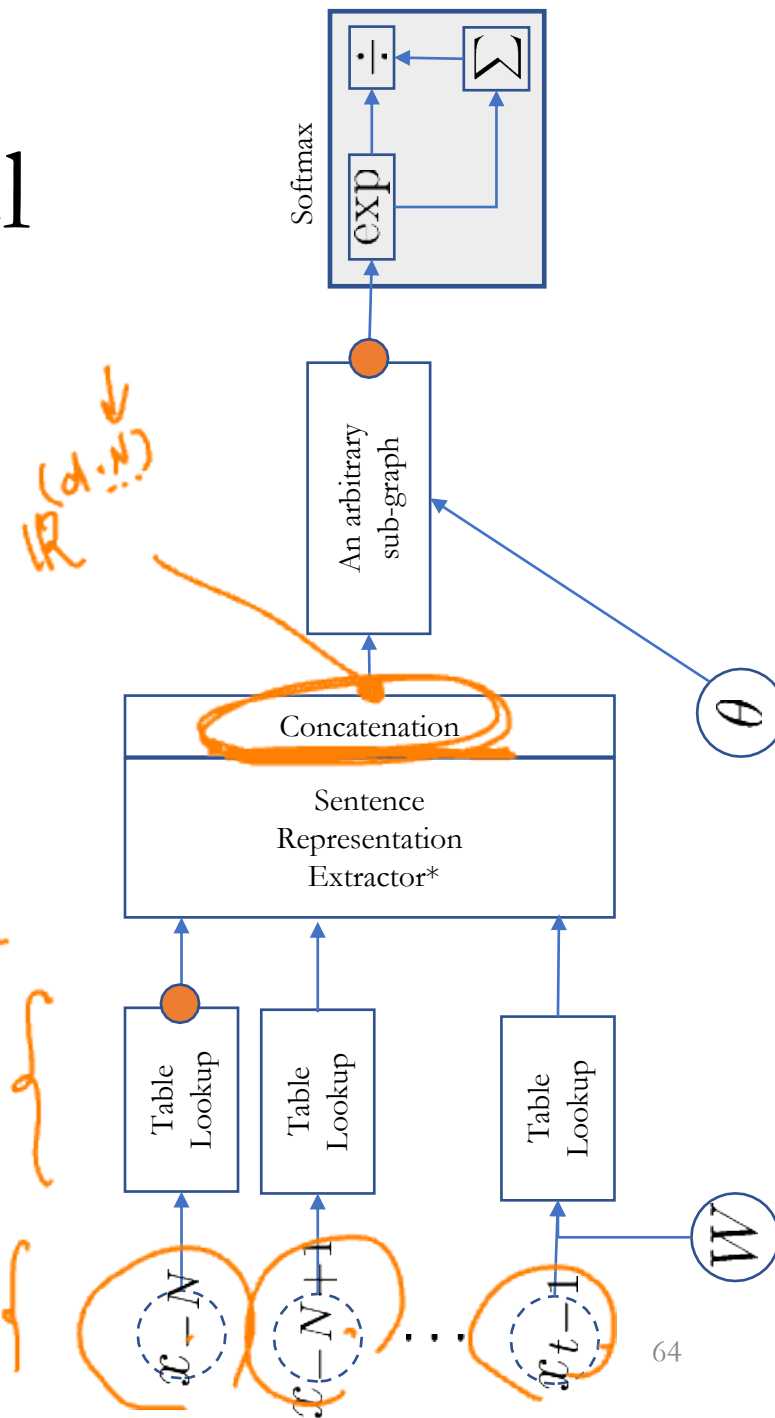
Neural N-Gram Language Model

- In practice,

1. Collect all n-grams from the corpus.
2. Shuffle all the n-grams to build a training set
3. Train the neural n-gram language model using stochastic gradient descent on minibatches containing 100-1000 n-grams.
4. Early-stop based on the validation set.
5. Report perplexity on the test set.

$$\text{ppl} = b^{\frac{1}{|D|} \sum_{(x_1, \dots, x_N) \in D} \log_b p(x_N | x_1, \dots, x_{N-1})}$$

...



Infinite context $n \rightarrow \infty$

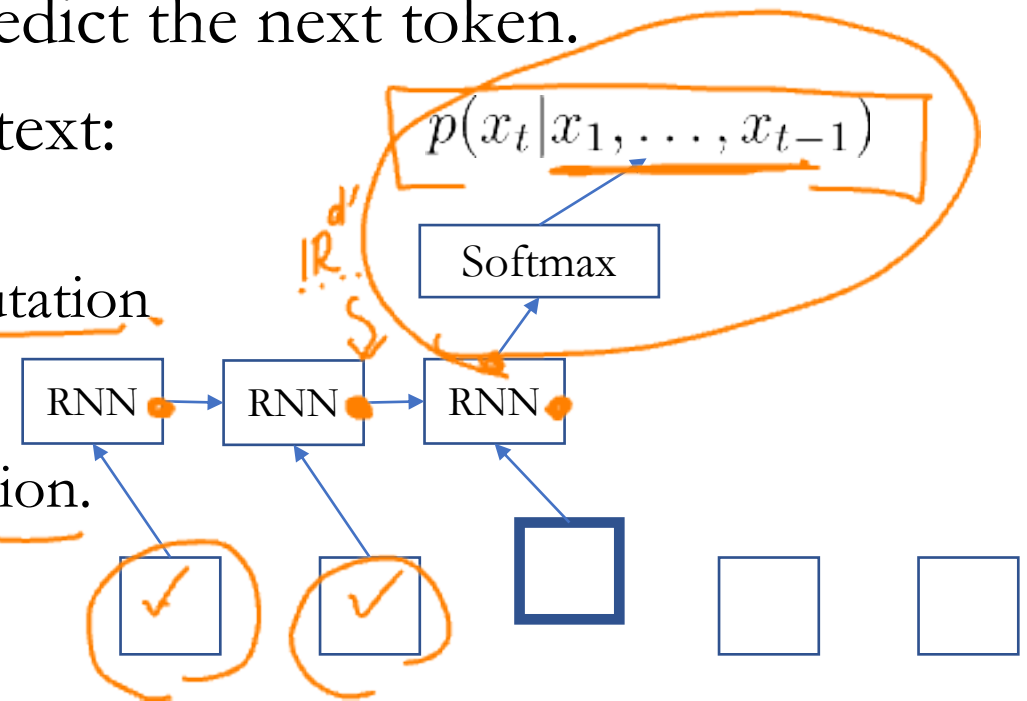
– CBoW Language Models

- Equivalent to the neural LM after replacing “concat” with “average”
 - “Averaging” allows the model to consider the infinite large context window.
- Extremely efficient, but a weak language model
 - Ignores the order of the tokens in the context windows.
 - Any language with a fixed order cannot be modelled well.
 - Averaging ignores the absolute counts, which may be important:
 - If the context window is larger, “verb” becomes less likely in SVO languages.

Infinite context $n \rightarrow \infty$

– Recurrent Language Models [Mikolov et al., 2010]

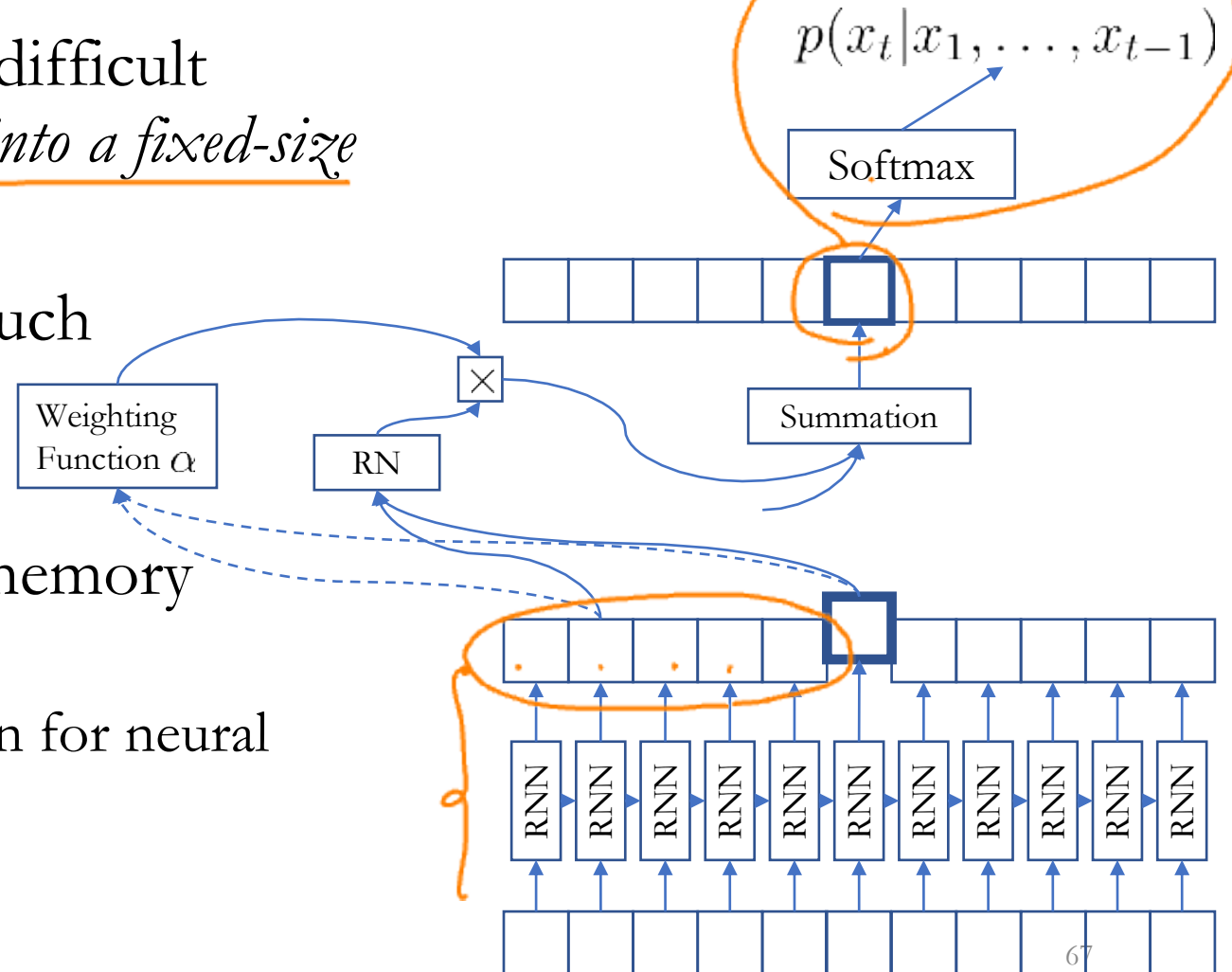
- A recurrent network summarizes all the tokens so far.
- Use the recurrent network's memory to predict the next token.
- Efficient online processing of a streaming text:
 - Constant time per step.
 - Constant memory throughout forward computation.
- Useful in practice:
 - Useful for autocomplete and keyword suggestion.
 - Scoring partial hypotheses in generation.



Infinite context $n \rightarrow \infty$

– Recurrent Memory Networks [Tran et al., 2016]

- The **recurrent network** solves a difficult problem: compress the entire context into a fixed-size memory vector.
- **Self-attention** does not require such compression but still can capture long-term dependencies.
- Combine these two: a recurrent memory network (RMN) [Tran et al., 2016]
 - RNMT+: a similar, recent extension for neural machine translation



In this lecture, we learned

- What autoregressive language modelling is:

$$p(X) = p(x_1)p(x_2|x_1) \cdots p(x_T|x_1, \dots, x_{T-1})$$

- How autoregressive language modelling transforms unsupervised learning into a series of supervised learning: - self-supervised lang.
 - It is a series of predicting the next token given previous tokens.
- How neural language modelling improves upon n-gram language models:
 - Continuous vector space facilitates generalization to unseen n-grams.
 - Infinitely large context window
- How sentence representation extraction is used for language modelling:
 - Convolutional language models, recurrent language models and self-attention language models..